

# Table

The `DataGridComponent` provides a comprehensive data table solution with advanced features including sorting, filtering, custom cell templates, and responsive design. It uses a flexible column definition system with content projection for maximum customization.

## How to use

Import the component and its directives, then define your table structure with custom templates.

## Basic Usage

Simple table with basic data display and column definitions.

```
import {  
  AavaDataGridComponent,  
  AavaColumnDefDirective,  
  AavaHeaderCellDefDirective,  
  AavaCellDefDirective,  
  AavaTagComponent,  
} from "@aava/play-core";
```

## Sorting

Table with sortable columns and visual sort indicators.

```

<div class="demo-page">
  <!-- Demo Content -->
  <div class="demo-content">
    <div class="container">
      <!-- Employee Table Section -->
      <div class="demo-section">
        <div class="table-container">
          <aava-data-grid
            [dataSource]="basicData"
            [displayedColumns]="displayedColumns"
            class="styled-data-grid"
          >
            <ng-container avaColumnDef="name">
              <ng-container *avaHeaderCellDef>
                <div class="header-cell">
                  <span class="header-text">Employee Name</span>
                </div>
              </ng-container>
              <ng-container *avaCellDef="let row">
                <div class="data-cell name-cell">
                  <span class="employee-name">{{ row.name }}</span>
                </div>
              </ng-container>
            </ng-container>

            <ng-container avaColumnDef="email">
              <ng-container *avaHeaderCellDef>
                <div class="header-cell">
                  <span class="header-text">Email Address</span>
                </div>
              </ng-container>
              <ng-container *avaCellDef="let row">
                <div class="data-cell email-cell">
                  <span class="email-text">{{ row.email }}</span>
                </div>
              </ng-container>
            </ng-container>

            <ng-container avaColumnDef="department">
              <ng-container *avaHeaderCellDef>
                <div class="header-cell">
                  <span class="header-text">Department</span>
                </div>
              </ng-container>
              <ng-container *avaCellDef="let row">
                <div class="data-cell department-cell">
                  <span class="department-badge">{{ row.department }}</span>
                </div>
              </ng-container>
            </ng-container>

            <ng-container avaColumnDef="status">
              <ng-container *avaHeaderCellDef>

```

```

        <div class="header-cell">
            <span class="header-text">Status</span>
        </div>
    </ng-container>
    <ng-container *avaCellDef="let row">
        <div class="data-cell status-cell">
            <aava-tag
                [label]="row.status"
                [color]="getStatusColor(row.status)"
                size="sm"
            ></aava-tag>
        </div>
    </ng-container>
</ng-container>
</aava-data-grid>
</div>
</div>
</div>
</div>

```

## Filtering

Advanced filtering capabilities with multiple filter conditions and operators.

```

basicData = [
  {
    id: 1,
    name: 'Alice Johnson',
    email: 'alice.johnson@example.com',
    department: 'Engineering',
    status: 'Active',
  },
  {
    id: 2,
    name: 'Bob Smith',
    email: 'bob.smith@example.com',
    department: 'Marketing',
    status: 'Active',
  },
  {
    id: 3,
    name: 'Carlos Martinez',
    email: 'carlos.martinez@example.com',
    department: 'Sales',
    status: 'Pending',
  },
  {
    id: 4,
    name: 'Diana Lee',
    email: 'diana.lee@example.com',
    department: 'Engineering',
    status: 'Inactive',
  },
  {
    id: 5,
    name: 'Ethan Brown',
    email: 'ethan.brown@example.com',
    department: 'HR',
    status: 'Active',
  },
];

displayedColumns = ['name', 'email', 'department', 'status'];

/**
 * Get the appropriate color for status tags
 */
getStatusColor(
  status: string
): 'success' | 'warning' | 'error' | 'info' | 'default' {
  switch (status.toLowerCase()) {
    case 'active':
      return 'success';
    case 'pending':
      return 'warning';
    case 'inactive':
      return 'error';
  }
}

```

```
    default:  
        return 'default';  
    }  
}
```

## Features

### Flexible Column System

- Content projection-based column definitions
- Custom header and cell templates
- Configurable sorting and filtering per column
- Dynamic column visibility

### Advanced Sorting

- Multi-column sorting support
- Visual sort indicators (ascending/descending)
- Configurable sort behavior per column
- Sort state management

### Powerful Filtering

- Multiple filter conditions and operators
- Real-time filtering with search
- Filter panel with advanced options
- Clear and apply filter actions

### Custom Templates

- Flexible cell content templates
- Custom header templates
- Template context with row data and index
- Support for complex cell content

### Responsive Design

- Horizontal scrolling for wide tables
- Mobile-friendly design
- Adaptive column sizing
- Touch-optimized interactions

## Performance Optimized

- OnPush change detection strategy
- Efficient data handling
- Optimized rendering
- Memory management

## API Reference

### Inputs

Property	Type	Default	Description
dataSource	any[]	[]	Array of data objects to display in the table
displayedColumns	string[]	[]	Array of column names to display

### Outputs

Property	Type	Description
dataSorted	EventEmitter<any[]>	Emitted when data is sorted with sorted data

## Directives

### AvaColumnDefDirective

Property	Type	Default	Description
avaColumnDef	string	-	Column name/identifier (required)
sortable	boolean	false	Enable sorting for this column

Property	Type	Default	Description
filter	boolean	false	Enable filtering for this column

## AvaHeaderCellDefDirective

Property	Type	Description
Template	TemplateRef<any>	Template for custom header cell content

## AvaCellDefDirective

Property	Type	Description
Template	TemplateRef<any>	Template for custom cell content with context

## Interfaces

## Methods

Method	Parameters	Description
onSort()	column: AvaColumnDefDirective	Handle column sorting
applySort()	None	Apply current sort to data
applyFilter()	columnName: string, event: Event	Apply filter to specific column
clearFilter()	columnName: string, event: any	Clear filter for specific column
openPanel()	columnName: string, event: any	Open filter panel for column
checkForOpen()	columnName: string	Check if filter panel is open for column

## Properties

Property	Type	Description
sortColumn	string   null	Currently sorted column
sortDirection	'asc'   'desc'   ''	Current sort direction
sortedData	any[]	Currently sorted and filtered data
filterColumn	Array<{column: string, type: string, value: any, open: boolean}>	Active filters
defaultFilterConditions	FilterCondition[]	Available filter conditions

## CSS Custom Properties

The component uses CSS custom properties for dynamic styling:

## Container Properties

Property	Description
--grid-font-family-body	Font family for table content
--grid-text-color	Text color for table content
--grid-background-color-odd	Background color for odd rows
--grid-background-color-even	Background color for even rows
--grid-border	Border color for grid elements

## Table Properties

Property	Description
--table-border	Border color for table elements

## CSS Classes

The component uses CSS classes for styling and state management:



## Container Classes

Class	Description
.ava-data-table-wrapper	Main table container
.data-table-wrapper	Inner table wrapper with scrolling
.ava-data-table	Main table element

## Cell Classes

Class	Description
.cell-wrapper	Header cell content wrapper
.grid-column-container	Column header container
.filter	Filter icon container
.filter-wrapper	Filter panel container
.default-filter-actions	Filter action buttons container
.cell-link	Link styling within cells

## State Classes

Class	Description
.sort-icon	Sort indicator icon
Various pseudo-classes	Hover and focus states

## Best Practices

### Data Structure

- Use consistent data structure across all rows
- Ensure column names match displayedColumns array
- Provide meaningful default values for missing data
- Optimize data for sorting and filtering

### Column Definitions

- Use descriptive column names
- Enable sorting only for relevant columns
- Enable filtering for searchable data
- Provide meaningful header labels

## **Custom Templates**

- Keep cell templates simple and focused
- Use template context for row data access
- Implement proper error handling in templates
- Consider accessibility in custom content

## **Performance**

- Limit data size for optimal performance
- Use OnPush change detection strategy
- Implement virtual scrolling for large datasets
- Optimize filter and sort operations

## **Accessibility**

- Provide proper ARIA labels
- Ensure keyboard navigation support
- Use semantic HTML structure
- Maintain color contrast ratios

## **Responsive Design**

- Test table on various screen sizes
- Implement horizontal scrolling for wide tables
- Consider mobile-specific interactions
- Optimize touch targets for mobile

## **Accessibility**

### **ARIA Support**

- Proper table semantics
- Sort and filter announcements
- Screen reader friendly navigation
- Status updates for dynamic content

## Keyboard Navigation

- Tab navigation through table elements
- Arrow key navigation between cells
- Enter/Space activation for actions
- Escape key for closing panels

## Focus Management

- Clear focus indicators
- Logical tab order
- Focus restoration after actions
- Focus trapping in modals/panels

## Screen Reader Support

- Descriptive labels for actions
- Context information for data
- Status announcements
- Clear navigation structure

## Browser Support

- Modern Browsers : Full support for all features
- CSS Grid/Flexbox : Required for layout
- ES6+ Features : Required for component functionality
- Template Ref : Required for content projection
- Change Detection : OnPush strategy support